
AWS Tools for Windows PowerShell

User Guide

Version 1.0



AWS Tools for Windows PowerShell: User Guide

Copyright © 2015 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Table of Contents

What are the AWS Tools for Windows PowerShell?	1
How to Use this Guide	1
Getting Set Up	2
Getting Setup	2
Prerequisites	2
Download and Install the PowerShell Tools	3
Enable Script Execution	3
Configure a PowerShell Console to Use the Tools	4
Versioning	5
AWS Account and Access Keys	6
See Also	7
Getting Started	8
AWS Credentials	8
Managing Profiles	9
Specifying Credentials	10
Credentials Search Order	11
AWS Regions	11
Cmdlets Discovery and Aliases	13
Cmdlet Discovery	13
Cmdlet Naming and Aliases	14
Pipelining and \$AWSHistory	15
\$AWSHistory	15
See Also	18
Using the AWS Tools for Windows PowerShell	19
Amazon EC2	20
Create a Key Pair	20
Create a Security Group	22
Find an AMI	26
Launch an Instance	28
Amazon S3	31
Create an Amazon S3 Bucket, Verify Its Region, and Optionally Remove It	31
Configure an Amazon S3 Bucket as a Website and Enable Logging	32
Upload Objects to an Amazon S3 Bucket	33
Delete Amazon S3 Objects and Buckets	34
Upload In-Line Text Content to Amazon S3	35
AWS Identity and Access Management	36
Create New IAM Users and Groups	36
Set an IAM Policy for an IAM User	37
Set an Initial Password for an IAM User	38
Create Security Credentials for an IAM User	38
Amazon SNS and SQS	39
Create an Amazon SQS queue and get queue ARN	39
Create an Amazon SNS topic	39
Give permissions to the SNS topic	40
Subscribe the queue to the SNS topic	40
Give permissions	41
Verify results	41
CloudWatch	42
Publish a Custom Metric to Your CloudWatch Dashboard	42
See Also	43
Document History	44
AWS Tools for Windows PowerShell v1.1.1.0	44
AWS Tools for Windows PowerShell v1.0.1.0	44
AWS Tools for Windows PowerShell v1.0.0.0	45

What are the AWS Tools for Windows PowerShell?

The AWS Tools for Windows PowerShell are a set of PowerShell cmdlets that are built on top of the functionality exposed by the AWS SDK for .NET. The AWS Tools for Windows PowerShell enable you to script operations on your AWS resources from the PowerShell command line. Although the cmdlets are implemented using the service clients and methods from the SDK, the cmdlets provide an idiomatic PowerShell experience for specifying parameters and handling results. For example, the cmdlets for the PowerShell Tools support PowerShell pipelining—that is, you can pipeline PowerShell objects both into and out of the cmdlets.

The AWS Tools for Windows PowerShell are flexible in how they enable you to handle credentials including support for the AWS Identity and Access Management (IAM) infrastructure; you can use the tools with IAM user credentials, temporary security tokens, and IAM roles.

The AWS Tools for Windows PowerShell support the same set of services and regions as supported by the SDK.

How to Use this Guide

The guide is divided into the following major sections:

[Setting up the AWS Tools for Windows PowerShell \(p. 2\)](#)

This section explains how to install the AWS Tools for Windows PowerShell. It also covers how to sign up for AWS if you don't already have an account. (An AWS account is required in order to use the PowerShell Tools.)

[Getting Started with the AWS Tools for Windows PowerShell \(p. 8\)](#)

This section describes the fundamentals of using the tools, such as specifying credentials and regions, finding cmdlets for a particular service, and using aliases for cmdlets.

[Using the AWS Tools for Windows PowerShell \(p. 19\)](#)

This section includes information about using the AWS Tools for Windows PowerShell to perform common AWS tasks.

Setting up the AWS Tools for Windows PowerShell

Setting up the AWS Tools for Windows PowerShell

Topics

- [Prerequisites \(p. 2\)](#)
- [Download and Install the AWS Tools for Windows PowerShell \(p. 3\)](#)
- [Enable Script Execution \(p. 3\)](#)
- [Configure a PowerShell Console to Use the AWS Tools for Windows PowerShell \(p. 4\)](#)
- [Versioning \(p. 5\)](#)

Prerequisites

To use the AWS Tools for Windows PowerShell, you must have an AWS account. If you do not yet have an AWS account, see [AWS Account and Access Keys \(p. 6\)](#) for instructions on how to sign up.

To use the AWS Tools for Windows PowerShell, your installed system must meet the following prerequisites:

- Microsoft Windows XP or later
- Windows PowerShell 2.0 or later

Windows 7 and Windows Server 2008 R2 come with Windows PowerShell 2.0 installed. Windows 8 and Windows Server 2012 come with Windows PowerShell 3.0 installed. For earlier versions of Windows, such as Windows XP, Windows Vista, Windows Server 2003, and Windows Server 2008, you can get PowerShell 2.0 by installing the Windows Management Framework

- [Windows Management Framework \(Windows PowerShell 2.0, WinRM 2.0, and BITS 4.0\)](#)

Download and Install the AWS Tools for Windows PowerShell

The AWS Tools for Windows PowerShell is one of the optional components that you can install using the AWS Tools for Windows installer .msi. Download the installer by navigating to the following URL and clicking **AWS Tools for Windows**.

<http://aws.amazon.com/powershell/>

The installer for the PowerShell Tools installs the most recent version of the [AWS SDK for .NET](#). If you have Microsoft Visual Studio installed, the installer can also install the [AWS Toolkit for Visual Studio](#).

All Windows Amazon Machine Images (AMIs) have the AWS Tools for Windows PowerShell pre-installed. For an example of using the PowerShell Tools on an Amazon EC2 instance, see the following sample in the AWS SDK for .NET.

New | Project... | AWS | Compute and Networking | AWS EC2 Sample

Enable Script Execution

In order to load the AWS Tools for Windows PowerShell module, you need to enable PowerShell script execution—if you have not already done so. To enable script execution, use the Set-ExecutionPolicy cmdlet to set a policy of RemoteSigned. By default, PowerShell uses a policy of Restricted. For more information about execution policies, see [Microsoft's TechNet documentation](#).

To enable script execution

1. Administrator privileges are required to set the execution policy. If you are not logged in as a user with administrator privileges, you need to open a PowerShell prompt as Administrator: Click **Start** and then click **All Programs**; click **Accessories**, and then click **Windows PowerShell**; now right-click **Windows PowerShell**, and select **Run as administrator** from the context menu.
2. At the command prompt, type: Set-ExecutionPolicy RemoteSigned

Note

On a 64-bit system, you will also need to perform these steps for the 32-bit version of PowerShell, **Windows PowerShell (x86)**.

If you do not have the execution policy set correctly, PowerShell generates the following message.

```
File C:\Users\teslan\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1
cannot be loaded because the execution
of scripts is disabled on this system. Please see "get-help about_signing" for
more details.
At line:1 char:2
+ . <<<< 'C:\Users\teslan\Documents\WindowsPowerShell\Microsoft.PowerShell_pro
file.ps1'
    + CategoryInfo          : NotSpecified: (:) [], PSSecurityException
    + FullyQualifiedErrorId : RuntimeException
```

The installer for the PowerShell Tools updates the [PSModulePath](#) to include the location of the directory that contains the AWSPowerShell module. As a result, if you are using PowerShell version 3, the

AWSPowerShell module is loaded automatically whenever you run one of the AWS cmdlets. This enables you to use the AWS cmdlets interactively even if the execution policy on your system is set to disallow script execution.

Because the PSModulePath includes the location of the AWS module's directory, the `Get-Module -ListAvailable` cmdlet shows the module.

```
PS C:\> Get-Module -ListAvailable
```

ModuleType	Name	ExportedCommands
Manifest	AppLocker	{}
Manifest	BitsTransfer	{}
Manifest	PSDiagnostics	{}
Manifest	TroubleshootingPack	{}
Manifest	AWSPowerShell	{Update-EBApplicationVersion, Set-DPStatus, Remove-IAMGroupPol...}

Configure a PowerShell Console to Use the AWS Tools for Windows PowerShell

The installer creates a **Start Menu** group called, **Amazon Web Services**, which contains a shortcut called **Windows PowerShell for AWS**. For PowerShell version 2, this shortcut automatically imports the AWSPowerShell module and then runs the `Initialize-AWSDefaults` cmdlet. For PowerShell version 3, the AWSPowerShell module is loaded automatically whenever you run an AWS cmdlet. So, for PowerShell version 3 installations, the shortcut created by the installer only runs the `Initialize-AWSDefaults` cmdlet. For more information about `Initialize-AWSDefaults`, see [Using AWS Credentials \(p. 8\)](#).

The installer also creates an additional shortcut called **AWS Tools for Windows**, which opens a visual display of AWS resources for Windows developers.

If you use PowerShell version 3 or if you only use the shortcut installed by the installer, you do not need to configure a PowerShell window to use the AWS Tools for Windows PowerShell. However, if, for example, you use PowerShell version 2 with a specially configured PowerShell window and want to add support for the tools, you need to load the AWS module yourself.

How to Load the AWS Tools for Windows PowerShell Module (PowerShell version 2)

To load the PowerShell Tools module into your current session

1. Open a PowerShell prompt and type the following command:

```
PS C:> import-module "C:\Program Files (x86)\AWS Tools\PowerShell\AWSPowerShell\AWSPowerShell.psd1"
```

2. To verify that the module was loaded, type the following command:

```
PS C:> Get-Module
```

If you see an entry in the list named **AWSPowerShell** (see the following), then the PowerShell Tools module was loaded successfully.

ModuleType	Name	ExportedCommands
Binary	AWSPowerShell	{Update-EBApplicationVersion, Set-S3ACL...

Load AWS CLI for PowerShell Module into Every Session (PowerShell version 2)

To automatically load the AWSPowerShell module every time you start a PowerShell session, you can add it to your PowerShell profile. Note, however, that adding commands to your PowerShell profile can slow down the speed at which a PowerShell session starts up.

The PowerShell \$profile variable contains the full path to the text file that contains your PowerShell profile. This variable is available only in a PowerShell session; it is not one of the regular Windows environment variables. To view the value of this variable, use echo.

```
echo $profile
C:\Users\kncubq\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1
```

You can edit this file with any text editor, such as notepad.exe.

```
notepad $profile
```

You may need to create both the profile directory and the profile itself if they do not already exist.

Versioning

New versions of the PowerShell Tools will release periodically to support new AWS services and features. To tell what version of the tools you have installed, use the following command:

```
PS C:\> Get-AWSPowerShellVersion
AWS Tools for Windows PowerShell
Version 1.0.0.9
Copyright 2012 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Amazon Web Services SDK for .NET
Version 1.5.9.1
Copyright 2009-2012 Amazon.com, Inc. or its affiliates. All Rights Reserved.

This software includes third party software subject to the following copyrights:
- Logging from log4net, Apache License
[http://logging.apache.org/log4net/license.html]
```

You can also specify the -ListServices option to see a list of which AWS services are supported in the current version of the tools.

```
PS C:\> Get-AWSPowerShellVersion -ListServices
AWS Tools for Windows PowerShell
Version 1.0.0.9
Copyright 2012 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Amazon Web Services SDK for .NET
Version 1.5.9.1
Copyright 2009-2012 Amazon.com, Inc. or its affiliates. All Rights Reserved.

This software includes third party software subject to the following copyrights:
- Logging from log4net, Apache License
[http://logging.apache.org/log4net/license.html]
```

Service	Noun Prefix	Version
-----	-----	-----
Auto Scaling	AS	2011-01-01
Amazon CloudFront	CF	2012-07-01
AWS CloudFormation	CFN	2010-05-15
Amazon CloudSearch	CS	2011-02-01
Amazon CloudWatch	CW	2010-08-01
Amazon ElastiCache	EC	2012-03-09
Amazon Elastic Compute Cloud	EC2	2012-10-01
Elastic Load Balancing	ELB	2012-06-01
Amazon Elastic MapReduce	EMR	2009-03-31
AWS Identity and Access Management	IAM	2010-05-08
AWS Import/Export	IE	2010-06-01
AWS Route 53	R53	2012-02-29
Amazon Relational Database Service	RDS	2012-09-17
Amazon Simple Storage Service	S3	2006-03-01
Amazon Simple Email Service	SES	2010-12-01
AWS Storage Gateway	SG	2012-06-30
Amazon Simple Notification Service	SNS	2010-03-31
Amazon Simple Queue Service	SQS	2012-11-05
AWS Security Token Service	STS	2011-06-15

AWS Account and Access Keys

To access AWS, you will need to sign up for an AWS account.

To sign up for an AWS account

1. Open <http://www.amazonaws.cn/>, and then click **Sign Up**.
2. Follow the on-screen instructions.

Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to <http://www.amazonaws.cn> and clicking **My Account/Console**.

To get your access key ID and secret access key

Access keys consist of an access key ID and secret access key, which are used to sign programmatic requests that you make to AWS. If you don't have access keys, you can create them by using the AWS

Management Console. We recommend that you use IAM access keys instead of AWS root account access keys. IAM lets you securely control access to AWS services and resources in your AWS account.

Note

To create access keys, you must have permissions to perform the required IAM actions. For more information, see [Granting IAM User Permission to Manage Password Policy and Credentials](#) in *IAM User Guide*.

1. Open the [IAM console](#).
2. From the navigation menu, click **Users**.
3. Select your IAM user name.
4. Click **User Actions**, and then click **Manage Access Keys**.
5. Click **Create Access Key**.

Your keys will look something like this:

- Access key ID example: AKIAIOSFODNN7EXAMPLE
- Secret access key example: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

6. Click **Download Credentials**, and store the keys in a secure location.

Your secret key will no longer be available through the AWS Management Console; you will have the only copy. Keep it confidential in order to protect your account, and never email it. Do not share it outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon will ever ask you for your secret key.

Related topics

- [What Is IAM?](#) in *IAM User Guide*
- [AWS Security Credentials](#) in *AWS General Reference*

See Also

- [Getting Started with the AWS Tools for Windows PowerShell](#) (p. 8)
- [Using the AWS Tools for Windows PowerShell](#) (p. 19)

Getting Started with the AWS Tools for Windows PowerShell

This section describes fundamentals of using the PowerShell Tools. For example, it explains how to specify which credentials and region the tools should use when interacting with AWS. This section also provides guidance on how to use standard PowerShell cmdlets such as `Get-Command` to discover AWS cmdlets.

Topics

- [Using AWS Credentials \(p. 8\)](#)
- [Specifying AWS Regions \(p. 11\)](#)
- [Cmdlet Discovery and Aliases \(p. 13\)](#)
- [Pipelining and \\$AWSHistory \(p. 15\)](#)

Using AWS Credentials

Each PowerShell Tools command must include a set of AWS credentials, which are used to cryptographically sign the corresponding web service request. You can specify credentials per-command, per-session, or for all sessions. To avoid accidentally exposing your credentials, you should not put literal credentials in a command. Instead, you can create a profile for each set of credentials that you want to use, and store the profile in either of two credentials stores. You then reference the appropriate profile by name in your command, and PowerShell Tools retrieves the associated credentials. For a general discussion of how to safely manage AWS credentials, see [Best Practices for Managing AWS Access Keys](#).

Note

If you do not yet have an AWS account, you will need one in order to obtain credentials and use the PowerShell Tools. For information about how to sign up for an account, see [AWS Account and Access Keys \(p. 6\)](#).

Topics

- [Managing Profiles \(p. 9\)](#)
- [Specifying Credentials \(p. 10\)](#)
- [Credentials Search Order \(p. 11\)](#)

Managing Profiles

PowerShell Tools can use either of two credentials stores.

- The SDK Store, which encrypts your credentials and stores them under your home folder.

The SDK Store can also be used by the [AWS SDK for .NET](#) and [AWS Toolkit for Visual Studio](#).

- The credentials file, which is also located under your home folder, but stores credentials as plaintext.

The credentials file can also be used by all AWS SDKs and the AWS Command Line Interface.

This topic describes how to use PowerShell Tools to manage your profiles in the SDK Store. You can also manage the SDK Store by using the [Toolkit for Visual Studio](#) or programmatically by using the [AWS SDK for .NET](#). For directions on how to manage profiles in the credentials file, see <http://docs.amazonaws.cn/general/latest/gr/aws-access-keys-best-practices.html>

Add a new profile

To add a new profile to the SDK Store, call `Set-AWSCredentials` as follows:

```
Set-AWSCredentials -AccessKey AKIAIOSFODNN7EXAMPLE -SecretKey wJalrXUtn  
FEMI/K7MDENG/bPxrFiCYEXAMPLEKEY -StoreAs MyProfileName
```

The parameters are:

- `-AccessKey` – The access key.
- `-SecretKey` – The secret key.
- `-StoreAs` – The profile name, which must be unique.

To specify the default profile, set the profile name to `default`.

Update a profile

The SDK Store must be maintained manually. If you later change credentials on the service—for example, by using the [IAM Console](#)—running a command with the locally stored credentials will fail with the following error message:

```
The AWS Access Key Id you provided does not exist in our records.
```

You can update a profile by repeating the `Set-AWSCredentials` command for the profile, and passing it the new access and secret keys.

List profiles

You can check the current list of names as follows:

```
Get-AWSCredentials -ListStoredCredentials
```

Remove a profile

To remove a profile, use the following command:

```
Clear-AWSCredentials -StoredCredentials MyProfileName
```

The `-StoredCredentials` parameter specifies the profile name.

Specifying Credentials

There are several ways to specify credentials. The preferred approach is to use a profile rather than incorporating literal credentials into your command line. The PowerShell Tools locates the profile using a search order that is described in [Credentials Search Order \(p. 11\)](#). This section describes the most common ways to specify a profile.

Note

You use the `-ProfileName` parameter to specify a profile. This parameter is equivalent to the `-StoredCredentials` parameter used by earlier PowerShell Tools releases. For backward compatibility, `-StoredCredentials` is still supported.

Default profile (recommended)

Use `Initialize-AWSDefaults` to specify a default profile for every PowerShell session.

```
PS C:\> Initialize-AWSDefaults -ProfileName MyProfileName -Region us-west-2
```

Note

The default credentials are included in the SDK Store under the default profile name. The command overwrites any existing profile with that name.

Session profile

Use `Set-AWSCredentials` to specify a default profile for a particular session. This profile overrides any default profile for the duration of the session.

```
PS C:> Set-AWSCredentials -ProfileName MyProfileName
```

The `-ProfileName` parameter specifies the profile name.

Note

In versions of the PowerShell Tools prior to v1.1, the above command did not work correctly, and would *overwrite* the profile specified by *MyProfileName*. We recommend using a more recent version of the PowerShell Tools.

Command profile

Use the `-ProfileName` parameter to specify a profile for a particular command. This profile override any default or session profiles. For example:

```
PS C:> Get-EC2Instance -ProfileName MyProfileName
```

Tip

When you specify a default or session profile, you can also include a `-Region` parameter to specify a default or session region. For more information, see [Specifying AWS Regions \(p. 11\)](#). The following example specifies a default profile and region.

```
PS C:\> Initialize-AWSDefaults -ProfileName MyProfileName -Region us-west-2
```

By default, the credentials file is assumed to be in its default location under the user's home folder (C:\users*username*\.aws\credentials). To specify a credentials file in an arbitrary location, include a `-ProfilesLocation` parameter, set to the credentials file path. For example, the following example specifies a non-default credentials file for a particular command.

```
PS C:> Get-EC2Instance -ProfileName MyProfileName -ProfileLocation C:\aws_ser  
vice_credentials\credentials
```

Credentials Search Order

When you run a command, PowerShell Tools searches for credentials in the following order and uses the first available set.

1. Use literal credentials that are embedded in the command line.

We strongly recommend using profiles rather than putting literal credentials in your command lines.

2. Use a specified profile name or profile location.

- If you specify only a profile name, use a specified profile from the SDK Store and, if that does not exist, the specified profile from the credentials file in the default location.
- If you specify only a profile location, use the default profile from that credentials file.
- If you specify a name and a location, use the specified profile from that credentials file.

If the specified profile or location is not found, the command throws an exception. Search proceeds to the following steps only if you have not specified a profile or location.

3. Use credentials specified by the `-Credentials` parameter.
4. Use a session profile.
5. Use a default profile, in the following order:
 - a. The default profile in the SDK store.
 - b. The default profile in the credentials file.
 - c. Use the AWS PS Default profile in the SDK Store.
6. If you are using running the command on an Amazon EC2 instance that is configured for an IAM role, use EC2 instance credentials stored in an instance profile.

For more information about using IAM roles for Amazon EC2 Instances, go to the [AWS Developer Guide for .NET](#).

If this search fails to locate the specified credentials, the command throws an exception.

Specifying AWS Regions

There are two ways to specify the AWS Region to use when running AWS CLI commands, the `-Region` common parameter, or the `Set-AWSDefaultRegion` command.

Most AWS cmdlets will fail if you do not specify a region. The exceptions are: [Amazon S3 \(p. 31\)](#), Amazon SES, and [AWS Identity and Access Management \(IAM\) \(p. 36\)](#).

In the absence of a specified region, Amazon S3 uses [US Standard](#) and Amazon SES uses [US East 1](#).

[Amazon SES](#) and [IAM](#) are services that do not require a region to be specified.

To specify the region for a single AWS command

- Add the `-Region` parameter to your command, such as:

```
PS C:> Get-EC2Image -Region us-west-1
```

To set a default region for all AWS CLI commands in the session

- From the PowerShell command prompt, type the following command:

```
PS C:> Set-DefaultAWSRegion us-west-1
```

Note

This setting will last only for the current session. To apply to all of your PowerShell sessions, you can add this command to your PowerShell profile as you did for the `import-module` command.

To view the current default region for all AWS CLI commands

- From the PowerShell command prompt, type the following command:

```
PS C:> Get-DefaultAWSRegion
```

SystemName	Name	IsShellDefault
-----	----	-----
us-west-1	US West (N. California)	True

To clear the current default region for all AWS CLI commands

- From the PowerShell command prompt, type the following command:

```
PS C:> Clear-DefaultAWSRegion
```

To view a list of all available AWS regions

- From the PowerShell command prompt, type the following command. Note that the third column identifies which region is the default for your current session.

```
PS C:> Get-AWSRegion
```

SystemName	Name	IsShellDefault
-----	----	-----
us-east-1	US East (Virginia)	False
us-west-1	US West (N. California)	True
us-west-2	US West (Oregon)	False
eu-west-1	EU West (Ireland)	False
ap-northeast-1	Asia Pacific (Tokyo)	False
ap-southeast-1	Asia Pacific (Singapore)	False
ap-southeast-2	Asia Pacific (Sydney)	False
sa-east-1	South America (Sao Paulo)	False
us-gov-west-1	US GovCloud West (Oregon)	False

Cmdlet Discovery and Aliases

This section discusses which services are supported by the AWS Tools for Windows PowerShell, the set of cmdlets provided by the tools in support of those services, and alternative names (aliases) for accessing those services.

Cmdlet Discovery

You can use the PowerShell `Get-Command` cmdlet to obtain a list of all of the cmdlets that are provided by tools.

```
Get-Command -Module AWSPowerShell
```

The `Get-Command` cmdlet generates this list in alphabetical order. Therefore, the list of cmdlets is sorted by PowerShell verb rather than PowerShell noun.

The following scripts generates a list of the cmdlets sorted by the PowerShell nouns that correspond to the supported AWS services.

```
$services =  
"as",           # AutoScaling  
"cf[a-mo-z]",  # AWS CloudFront  
"cfn",         # AWS CloudFormation  
"cs[a-uw-z]",  # Amazon CloudSearch  
"cw",          # Amazon CloudWatch  
"eb",          # AWS Elastic Beanstalk  
"ec2",         # Amazon EC2  
"ec",          # Amazon ElastiCache  
"elb",         # Elastic Load Balancing  
"emr",         # Amazon Elastic Map Reduce  
"iam",         # AWS Identity and Access Management  
"ie",          # AWS Import/Export  
"r53",         # Amazon Route 53  
"rds",         # Amazon RDS  
"s3",          # Amazon S3  
"sg",          # Amazon Storage Gateway  
"ses",         # Amazon SES  
"sns",         # Amazon SNS  
"sqs",         # Amazon SQS  
"sts",         # AWS STS  
  
foreach ($s in $services) { "-----"; Get-Command -noun ${s}* }
```

You can also use the PowerShell `Select-String` cmdlet to filter the list of cmdlets returned by `Get-Command`. For example, to view the set of AWS cmdlets that work with regions, you could type the following command.

```
PS C:> Get-Command -Module AWSPowerShell | Select-String region  
  
Clear-DefaultAWSRegion  
Get-AWSRegion  
Get-DefaultAWSRegion  
Get-EC2Regions  
Set-DefaultAWSRegion
```

Cmdlet Naming and Aliases

The cmdlets provided by the PowerShell Tools for a given service correspond approximately to the methods provided by the SDK for that service. However, because of PowerShell's naming conventions, the name of a cmdlet may be somewhat different than the name of the corresponding method. For example, the `Get-EC2Instances` cmdlet performs a similar function to the Amazon EC2 `DescribeInstances` method.

In other cases, the cmdlet name may be similar to a method name, but it may actually perform a different function. For example, the Amazon S3 `GetObject` method retrieves an Amazon S3 object. However, the `Get-S3Object` cmdlet returns *information* about an Amazon S3 object rather than the object itself.

```
PS C:> Get-S3Object -BucketName text-content -Key text-object

Key           : text-object.txt
BucketName    : text-content
LastModified  : Mon, 27 Aug 2012 19:39:34 GMT
ETag         : "f738612c5e842b39819c6d8fc4eb5b9b"
Size          : 20622
Owner         : Amazon.S3.Model.Owner
StorageClass  : STANDARD
```

To retrieve the object with the PowerShell Tools, use the `Read-S3Object` cmdlet.

```
PS C:> Read-S3Object -BucketName text-content -Key text-object.txt -file
c:\tmp\text-object-download.txt

Mode                LastWriteTime         Length Name
----                -
-a---             11/5/2012   7:29 PM         20622 text-object-download.txt
```

Note

The cmdlet help for an AWS cmdlet provides the name of the SDK API that corresponds to the cmdlet. For more information about the standard PowerShell verbs and their expected meanings, go to the [Windows DevCenter](#).

All AWS cmdlets that use the `Remove` verb, and the `Stop-EC2Instance` cmdlet when used with the `-Terminate` switch, now prompt for confirmation before proceeding. To bypass confirmation, use the `-Force` switch.

The AWS cmdlets do not support the `-WhatIf` switch.

Aliases

The setup program for the PowerShell Tools installs an aliases file that contains aliases for many of the tools cmdlets. You may find these aliases to be more intuitive than the cmdlet names. For example, aliases are provided that are prefixed with the service name—rather than a PowerShell verb—and followed by an SDK method name. An example is the `EC2-DescribeInstances` alias.

Other aliases use verbs that, although they do not follow standard PowerShell conventions, may be more descriptive of the actual operation. For example, the alias file maps the alias `Get-S3Content` to the cmdlet `Read-S3Object`.

```
PS C:> Set-Alias -Name Get-S3Content -Value Read-S3Object
```

The aliases file is located in the AWS Tools for Windows PowerShell installation directory. To load the aliases into your environment, "dot-source" the file.

```
PS C:>. c:\Program Files (x86)\AWS Tools\PowerShell\AWSPowershell\AWSAliases.ps1
```

Pipelining and \$AWSHistory

For service calls that return collections, the objects within the collection are now always enumerated to the pipeline. Result objects that contain additional fields beyond the collection and which are not paging control fields have these fields added as Note properties for the calls. These Note properties are logged in the new \$AWSHistory session variable, should you need to access this data. The \$AWSHistory variable is described in the next section.

Note

In versions of the PowerShell Tools prior to v1.1, the collection object itself was emitted, which required the use of `foreach {$_ .GetEnumerator() }` to continue pipelining.

Examples

Return a collection of Amazon EC2 machine images (AMIs) across all regions.

```
Get-AWSRegion | % { Get-EC2Image -Owner self -Region $_ }
```

Stop all Amazon EC2 instances in the current default region.

```
Get-EC2Instance | Stop-EC2Instance
```

Because collections enumerate to the pipeline, the output from a given cmdlet might be \$null, a single object, or a collection. If it is a collection, you can use the `.Count` property to determine the size of the collection. However, the `.Count` property is not present when only a single object is emitted. If your script needs to determine, in a consistent way, how many objects were emitted, use the new `EmittedObjectsCount` property of the last command value in \$AWSHistory.

\$AWSHistory

To better support pipelining, output from AWS cmdlets is no longer reshaped to include the service (SDK) response and result instances as Note properties on the emitted collection object. Instead, for those calls that emit a single collection as output, the collection is now enumerated to the PowerShell pipeline as described in previously. This means that the SDK response/result data cannot exist in the pipe as there is no containing collection object to which it can be attached.

Although most users probably won't need this data, it can be useful for diagnostic purposes as you can see exactly what was sent to and received from the underlying AWS service call(s) made by the cmdlet.

Starting with version 1.1, this data and more is now available in a new shell variable named \$AWSHistory. This variable maintains a record of AWS cmdlet invocations and for each, the service responses that were received. Optionally, this history can be configured to also record the service requests that each cmdlet made. Additional useful data such as the overall execution time of the cmdlet can also be obtained from each entry.

Each entry in the \$AWSHistory.Commands list is of type `AWSCmdletHistory`. This type has the following useful members:

CmdletName

Name of the cmdlet.

CmdletStart

DateTime that the cmdlet was initialized by the shell.

CmdletEnd

DateTime that the cmdlet finished all processing.

Requests

If request recording is enabled, list of last service requests.

Responses

List of last service responses received.

LastServiceResponse

Helper to return the most recent service response.

LastServiceRequest

Helper to return the most recent service response, if available.

Note that the \$AWSHistory variable is not created until an AWS cmdlet making a service call is used. It evaluates to \$null until that point.

Note

Earlier versions of the PowerShell Tools emitted data related to services responses as Note properties on the returned object. These are now found on the response entries recorded for each invocation in the list.

Set-AWSHistoryConfiguration

A given cmdlet invocation can hold zero or more service request and response entries. To limit memory impact the \$AWSHistory list keeps a record of only the last 5 cmdlet executions by default and for each, the last 5 service responses (and if enabled, last 5 service requests). These default limits can be changed using the new Set-AWSHistoryConfiguration cmdlet. It allows you to both control the size of the list and whether service requests are also logged:

```
Set-AWSHistoryConfiguration -MaxCmdletHistory <value> -MaxServiceCallHistory  
<value> -RecordServiceRequests
```

The -MaxCmdletHistory parameter sets the maximum number of cmdlets that can be tracked at any time. A value of 0 turns off recording of AWS cmdlet activity. The -MaxServiceCallHistory parameter sets the maximum number of service responses (and/or requests) that are tracked for each cmdlet. The -RecordServiceRequests parameter, if specified, turns on tracking of service requests for each cmdlet. All parameters are optional.

If run with no parameters, Set-AWSHistoryConfiguration simply turns off any prior request recording, leaving the current list sizes unchanged.

To clear all entries in the current history list, use the new Clear-AWSHistory cmdlet.

\$AWSHistory Examples

Enumerate the details of the AWS cmdlets that are being held in the list to the pipeline.

```
PS C:\> $AWSHistory.Commands
```

Access the details of the last AWS cmdlet that was run:

```
PS C:\> $AWSHistory.LastCommand
```

Access the details of the last service response received by the last AWS cmdlet that was run. If an AWS cmdlet is paging output, it may make multiple service calls to obtain either all data or the maximum amount of data (determined by parameters on the cmdlet).

```
PS C:\> $AWSHistory.LastServiceResponse
```

Access the details of the last request made (again, a cmdlet may make more than one request if it is paging on the user's behalf). Yields \$null unless service request tracing is enabled.

```
PS C:\> $AWSHistory.LastServiceRequest
```

Automatic Page-to-Completion for Operations that Return Multiple Pages

For service APIs that impose a default maximum object return count for a given call or that support pageable result sets, all cmdlets "page-to-completion" by default; the cmdlet makes as many calls as necessary on your behalf to return the complete data set to the pipeline.

In the following example, which uses `Get-S3Object`, the `$c` variable contains `S3Object` instances for every key in the bucket `test`, potentially a very large data set.

```
$c = Get-S3Object -BucketName test
```

If you want to retain control of the amount of data returned, you can continue to use parameters on the individual cmdlets (e.g. `MaxKeys` on `Get-S3Object`) or you can explicitly handle paging yourself by using a combination of paging parameters on the cmdlets, and data placed in the `$AWSHistory` variable to get the service's next token data. The following example uses the `MaxKeys` parameter to limit the number of `S3Object` instances returned to no more than the first 500 found in the bucket.

```
$c = Get-S3Object -BucketName test -MaxKeys 500
```

To know if more data was available but not returned, use the `$AWSHistory` session variable entry that recorded the service calls made by the cmdlet.

If the following expression evaluates to `$true`, you can find the next marker for the next set of results using `$AWSHistory.LastServiceResponse.NextMarker`.

```
$AWSHistory.LastServiceResponse -ne $null && $AWSHistory.LastServiceResponse.IsTruncated
```

To manually control paging with `Get-S3Object`, use a combination of the `MaxKeys` and `Marker` parameters for the cmdlet and the `IsTruncated/NextMarker` notes on the last recorded response. In the following example, the variable `$c` contains up to a maximum of 500 `S3Object` instances for the next 500 objects that are found in the bucket after the start of the specified key prefix marker.

```
$c = Get-S3Object -BucketName test -MaxKeys 500 -Marker $AWSHistory.LastService  
Response.NextMarker
```

See Also

- [Using the AWS Tools for Windows PowerShell \(p. 19\)](#)

Using the AWS Tools for Windows PowerShell

This section provides examples of using the AWS Tools for Windows PowerShell to access AWS services. These examples are intended to demonstrate how to use the cmdlets to perform actual administrative tasks.

Note Regarding Returned Objects for the PowerShell Tools

In some cases, the object returned from an PowerShell Tools cmdlet does not mirror what is returned from the corresponding API in the AWS SDK for .NET. For example, `List-S3Buckets` emits a `Buckets` collection, not an Amazon S3 response object. Similarly, `Get-EC2Instance` emits a `Reservation` collection, not a `DescribeEC2Instances` result object. This behavior is by design and is intended to have the PowerShell Tools experience be more consistent with idiomatic PowerShell.

The actual service responses are stored in `note` properties on the returned objects and are therefore available if you need to access them. For API actions that support `NextToken` fields, these are also attached as `note` properties.

[Amazon EC2 \(p. 20\)](#)

This section walks through the steps required to launch an Amazon EC2 instance including how to:

- Retrieve a list of Amazon Machine Images (AMIs).
- Create a key pair.
- Create and configure a security group.
- Launch the instance and retrieve information about it.

[Amazon S3 \(p. 31\)](#)

The section walks through the steps required to create a static website hosted in Amazon S3. It demonstrates how to:

- Create and delete Amazon S3 buckets.
- Upload files to an Amazon S3 bucket as objects.
- Delete objects from an Amazon S3 bucket.
- Designate an Amazon S3 bucket as a website.

[AWS Identity and Access Management \(p. 36\)](#)

This section demonstrates basic operations in AWS Identity and Access Management (IAM) including how to:

- Create an IAM group.

- Create an IAM user.
- Add an IAM user to an IAM group.
- Specify a policy for an IAM user.
- Set a password and credentials for an IAM user.

[Amazon SNS and Amazon SQS \(p. 39\)](#)

This section walks through the steps required to subscribe an Amazon SQS queue to an Amazon SNS topic. It demonstrates how to:

- Create an Amazon SNS topic.
- Create an Amazon SQS queue.
- Subscribe the queue to the topic.
- Send a message to the topic.
- Receive the message from the queue.

[CloudWatch \(p. 42\)](#)

This section provides an example of how to publish custom data to CloudWatch.

- Publish a Custom Metric to Your CloudWatch Dashboard.

Amazon EC2 from the AWS Tools for Windows PowerShell

You can perform common tasks related to Amazon EC2 using the AWS Tools for Windows PowerShell.

The example commands shown here assume that you have set default credentials and a default region for your PowerShell session. Therefore, we don't include credentials or region when we invoke the cmdlets. For more information, see [Getting Started with the AWS Tools for Windows PowerShell \(p. 8\)](#).

Topics

- [Create an Amazon EC2 Key Pair Using Windows PowerShell \(p. 20\)](#)
- [Create a Security Group Using Windows PowerShell \(p. 22\)](#)
- [Find an Amazon Machine Image Using Windows PowerShell \(p. 26\)](#)
- [Launch an Amazon EC2 Instance Using Windows PowerShell \(p. 28\)](#)

Create an Amazon EC2 Key Pair Using Windows PowerShell

You must specify a key pair when you launch and connect to an Amazon EC2 instance. This documentation describes how to use the AWS Tools for Windows PowerShell to create and manage your key pairs.

Topics

- [Creating a Key Pair \(p. 21\)](#)
- [Viewing the Fingerprint of Your Key Pair \(p. 22\)](#)
- [Storing Your Private Key \(p. 22\)](#)
- [Removing Your Key Pair \(p. 22\)](#)

Creating a Key Pair

The following example uses the `New-EC2KeyPair` cmdlet to create a key pair. The returned object is stored in the PowerShell variable `$myPSKeyPair`

```
PS C:> $myPSKeyPair = New-EC2KeyPair -KeyName myPSKeyPair
```

Pipe the key pair object into the `Get-Member` cmdlet to view the object's members.

```
PS C:> $myPSKeyPair | Get-Member

TypeName: Amazon.EC2.Model.KeyPair

Name                MemberType Definition
----                -
Equals              Method    bool Equals(System.Object obj)
GetHashCode          Method    int GetHashCode()
GetType             Method    type GetType()
ToString            Method    string ToString()
KeyFingerprint       Property  System.String KeyFingerprint {get;set;}
KeyMaterial          Property  System.String KeyMaterial {get;set;}
KeyName             Property  System.String KeyName {get;set;}
```

Pipe the key pair object into the `Format-List` cmdlet to view values of the `KeyName`, `KeyFingerprint`, and `KeyMaterial` members. (The output has been truncated for readability.)

```
PS C:> $myPSKeyPair | Format-List KeyName, KeyFingerprint, KeyMaterial

KeyName           : myPSKeyPair
KeyFingerprint    : 09:06:70:8e:26:b6:e7:ef:8f:fe:4a:1d:bc:9c:6a:63:11:ac:ad:3c
KeyMaterial       : ----BEGIN RSA PRIVATE KEY----
                   MIIEogIBAAKCAQEAKK+ANYUS9c7niNjYfaCn6KYj/D0I6dJnFoQE...
                   Mz6btoxPcE7EMeH1wySU8nouAS9xbl9l7+Vkd74bN9KmNcPa/Mu...
                   Zyn4vVe0Q5il/MpkrRogHq0B0rigeTeV5Yc3lv00RFFPu0Kz4kcm...
                   w3Jg8dKswn0pl0px7V3sRC02KgJIbejQUvBFGi50QK9bm4tXBIEc...
                   daxKIAQMtDUdmBDRhR1/YMv8itFe5DiLLbq7Ga+FDcS85NstBa3h...
                   iuskGkcvgWkcFQkLmRHRoDpPb+OdFsZtjHZDpMVfMA9tT8EdbkEF...
                   3SrNeqZPsxJJIX0odb3CxLJpg75JU5kyWnb0+sDNVHoJiZCULCr0...
                   GGLfEgB95KjGIk7zEv2Q7K6s+DHclrDeMZwa7KFNRZuCuX7jssC...
                   x098abxMr3o3TNU6p1ZYRJEQ0oJr0W+kc+/8SWb8NIwfltwmJEy...
                   1BX9X8WFX/A8VLHrT1elrKmlKNECgYEAwltkv1p0JAFhz9p7ZFEv...
                   vvVsPaF0Ev9bk9pqhx269PB50x2KokwCagDMMaYvasWobuLmNu/1...
                   lmwRx7KTeQ7W1J30LgxHA1QNMkip9c4Tb3q9vVc3t/fPf8vwfJ8C...
                   63g6N6rk2FkHZX1E62BgbewUd3eZ0S05Ip4VUdvtGcuc8/qa+e5C...
                   KXgyt9nl64pMv+VaxfXkZhdLAdY0Khc9TGB9++VMsG5TrD15YJId...
                   gYALEI7m1jJKpHWAes0hiemw5VmKyIZpzGstSJsFSTERLAjiETDH...
                   YAtnI4J8dRyP9I7BOV0n3wNfIjk85gi1/00c+j8S65giLAFndWGR...
                   9R9wIkM5BMUcSRRCdy0yuwKBgEbkOnGGSD0ah4HkvrUkepIbUDTD...
                   AnEBM1cXI5UT7BfKInpUihZi59QhgdK/hk0SmWhlZGWikJ5VizBf...
                   drkBr/vTKVRMTi3LVFB7KkIV1xJxC5E/BZ+YdZEpWoCZAoGAC/Cd...
                   TTld5N6opgOXAcQJwzqoGa9ZMwc5Q9f4bfRc67emkw0ZAAwSsvWR...
                   x302duuy7/smTwWwskEWRK5IrUxoMv/VVYaqdzc0ajwieNrblr7c...
                   -----END RSA PRIVATE KEY-----
```

The `KeyMaterial` member stores the private key for the key pair. The public key is stored in AWS. You can't retrieve the public key from AWS, but you can verify the public key by comparing the `KeyFingerprint` for the private key to that returned from AWS for the public key.

Viewing the Fingerprint of Your Key Pair

You can use the `Get-EC2KeyPair` cmdlet to view the fingerprint for your key pair.

```
PS C:> Get-EC2KeyPair -KeyName myPSKeyPair | format-list KeyName, KeyFingerprint

KeyName           : myPSKeyPair
KeyFingerprint    : 09:06:70:8e:26:b6:e7:ef:8f:fe:4a:1d:bc:9c:6a:63:11:ac:ad:3c
```

Storing Your Private Key

To store the private key to a file, pipe the `KeyFingerMaterial` member to the `Out-File` cmdlet.

```
PS C:> $myPSKeyPair.KeyMaterial | Out-File -Encoding ascii myPSKeyPair.pem
```

You must specify `-Encoding ascii` when writing the private key to a file. Otherwise, tools such as `openssl` may not be able to read the file correctly. You can verify that the format of the resulting file is correct by using a command such as the following:

```
openssl rsa -check < myPSKeyPair.pem
```

(The `openssl` tool is not included with the AWS Tools for Windows PowerShell or the AWS SDK for .NET.)

Removing Your Key Pair

You'll need your key pair to launch and connect to an instance. When you have finished using a key pair, you can remove it. To remove the public key from AWS, use the `Remove-EC2KeyPair` cmdlet. When prompted, press **Enter** to remove the key pair.

```
PS C:> Remove-EC2KeyPair -KeyName myPSKeyPair

Remove-EC2KeyPair
Are you sure you want to remove keypair 'myPSKeyPair'?
[Y] Yes  [N]  [S] Suspend  [?] Help (default is "Y"):
```

The variable, `$myPSKeyPair`, still exists in the current PowerShell session and still contains the key pair information. The `myPSKeyPair.pem` file exists as well. However, the private key is no longer valid because the public key for the key pair is no longer stored in AWS.

Create a Security Group Using Windows PowerShell

You can use the AWS Tools for Windows PowerShell to create and configure a security group. When you create a security group, you specify whether it is for EC2-Classic or EC2-VPC. The response is the ID of the security group.

If you need to connect to your instance, you must configure the security group to allow SSH traffic (Linux) or RDP traffic (Windows).

Topics

- [Prerequisites \(p. 23\)](#)
- [Creating a Security Group for EC2-Classic \(p. 23\)](#)
- [Creating a Security Group for EC2-VPC \(p. 24\)](#)

Prerequisites

You need the public IP address of your computer, in CIDR notation. You can get the public IP address of your local computer using a service. For example, we provide the following service: <http://checkip.amazonaws.com/>. To locate another service that provides your IP address, use the search phrase "what is my IP address". If you are connecting through an ISP or from behind your firewall without a static IP address, you need to find the range of IP addresses used by client computers.

If you use 0.0.0.0/0, you enable all IP addresses to access your instance. For the SSH and RDP protocols, this is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, you'll authorize only a specific IP address or range of addresses to access your instance.

Creating a Security Group for EC2-Classic

The following example uses the `New-EC2SecurityGroup` cmdlet to create a security group for EC2-Classic.

```
PS C:> New-EC2SecurityGroup -GroupName myPSSecurityGroup -GroupDescription "EC2-Classic from PowerShell"
sg-9cf9e5d9
```

To view the initial configuration of the security group, use the `Get-EC2SecurityGroup` cmdlet.

```
PS C:> Get-EC2SecurityGroup -GroupNames myPSSecurityGroup

OwnerId           : 123456789012
GroupName         : myPSSecurityGroup
GroupId           : sg-9cf9e5d9
Description       : EC2-Classic from PowerShell
IpPermissions     : {}
IpPermissionsEgress : {}
VpcId             :
Tags              : {}
```

To configure the security group to allow inbound traffic on TCP port 22 (SSH) and TCP port 3389, use the `Grant-EC2SecurityGroupIngress` cmdlet. For example, here's how you enable SSH traffic from a single IP address, 203.0.113.25/32.

```
PS C:> $cidrBlocks = New-Object 'collections.generic.list[string]'
PS C:> $cidrBlocks.add("203.0.113.25/32")
PS C:> $ipPermissions = New-Object Amazon.EC2.Model.IpPermission -Property
@{IpProtocol = "tcp"
>> FromPort = 22
>> ToPort = 22
```

```
>> IpRanges = $cidrBlocks}
>>
PS C:> Grant-EC2SecurityGroupIngress -GroupName myPSSecurityGroup -IpPermissions
$ipPermissions
```

To verify the security group has been updated, use the `Get-EC2SecurityGroup` cmdlet again. Note that you can't specify an outbound rule for EC2-Classic.

```
PS C:> Get-EC2SecurityGroup -GroupNames myPSSecurityGroup

OwnerId           : 123456789012
GroupName         : myPSSecurityGroup
GroupId           : sg-9cf9e5d9
Description       : EC2-Classic from PowerShell
IpPermissions     : {Amazon.EC2.Model.IpPermission}
IpPermissionsEgress : {}
VpcId             :
Tags              : {}
```

To view the security group rule, use the `IpPermissions` property.

```
PS C:> (Get-EC2SecurityGroup -GroupNames myPSSecurityGroup).IpPermissions

IpProtocol      : tcp
FromPort        : 22
ToPort          : 22
UserIdGroupPairs : {}
IpRanges        : {203.0.113.25/32}
```

Creating a Security Group for EC2-VPC

The following example uses the `New-EC2SecurityGroup` cmdlet to create a security group for the specified VPC.

```
PS C:> $groupid = New-EC2SecurityGroup -VpcId "vpc-da0013b3" -GroupName
"myPSSecurityGroup" -GroupDescription "EC2-VPC from PowerShell"
```

To view the initial configuration of the security group, use the `Get-EC2SecurityGroup` cmdlet. By default, the security group for a VPC contains a rule that allows all outbound traffic. Notice that you can't reference a security group for EC2-VPC by name.

```
PS C:> Get-EC2SecurityGroup -GroupId sg-5d293231

OwnerId           : 123456789012
GroupName         : myPSSecurityGroup
GroupId           : sg-5d293231
Description       : EC2-VPC from PowerShell
IpPermissions     : {}
IpPermissionsEgress : {Amazon.EC2.Model.IpPermission}
VpcId             : vpc-da0013b3
Tags              : {}
```

To define the permissions for inbound traffic on TCP port 22 (SSH) and TCP port 3389, use the `New-Object` cmdlet, which works with PowerShell 2.0 and later. For example, here's how you define permissions for TCP ports 22 and 3389 from a single IP address, 203.0.113.25/32.

```
PS C:> $ip1 = new-object Amazon.EC2.Model.IpPermission
PS C:> $ip1.IpProtocol = "tcp"
PS C:> $ip1.FromPort = 22
PS C:> $ip1.ToPort = 22
PS C:> $ip1.IpRanges.Add("203.0.113.25/32")

PS C:> $ip2 = new-object Amazon.EC2.Model.IpPermission
PS C:> $ip2.IpProtocol = "tcp"
PS C:> $ip2.FromPort = 3389
PS C:> $ip2.ToPort = 3389
PS C:> $ip2.IpRanges.Add("203.0.113.25/32")
```

To configure the security group with the defined permissions, use the `Grant-EC2SecurityGroupIngress` cmdlet. For example:

```
PS C:> Grant-EC2SecurityGroupIngress -GroupId $groupid -IpPermissions @( $ip1,
    $ip2 )
```

To verify the security group has been updated, use the `Get-EC2SecurityGroup` cmdlet again.

```
PS C:> Get-EC2SecurityGroup -GroupIds sg-5d293231

OwnerId           : 123456789012
GroupName         : myPSSecurityGroup
GroupId           : sg-5d293231
Description       : EC2-VPC from PowerShell
IpPermissions     : {Amazon.EC2.Model.IpPermission}
IpPermissionsEgress : {Amazon.EC2.Model.IpPermission}
VpcId             : vpc-da0013b3
Tags              : {}
```

To view the inbound rules, use the `IpPermissions` property.

```
PS C:> ($groupid | Get-EC2SecurityGroup).IpPermissions

IpProtocol      : tcp
FromPort        : 22
ToPort          : 22
UserIdGroupPairs : {}
IpRanges        : {203.0.113.25/32}

IpProtocol      : tcp
FromPort        : 3389
ToPort          : 3389
UserIdGroupPairs : {}
IpRanges        : {203.0.113.25/32}
```

Find an Amazon Machine Image Using Windows PowerShell

When you launch an Amazon EC2 instance, you need to specify an Amazon Machine Image (AMI) for the instance configuration in which you are interested. However, the IDs for the AMIs that AWS provides to customers change periodically because AWS updates these images with new features and security enhancements. This section describes how to use the `Get-EC2Image` and `Get-EC2ImageByName` cmdlets to find an AMI and get its current ID.

Topics

- [Get-EC2Image](#) (p. 26)
- [Get-EC2ImageByName](#) (p. 27)

Get-EC2Image

The `Get-EC2Image` cmdlet retrieves a list of AMIs that you can use.

Use the `-Owner` parameter with the array value `amazon, self` so that `Get-EC2Image` retrieves only AMIs that belong to Amazon or to you. In this context, *you* refers to the user who corresponds to the credentials with which the cmdlet is invoked.

```
PS C:> Get-EC2Image -Owner amazon, self
```

You can scope the results using the `-Filter` parameter. To specify the filter, create an object of type `Amazon.EC2.Model.Filter`.

```
$platform_values = New-Object 'collections.generic.list[string]'  
PS C:> $platform_values.add("windows")  
PS C:> $filter_platform = New-Object Amazon.EC2.Model.Filter -Property @{Name  
= "platform"; Values = $platform_values}
```

Next, call the `Get-EC2Image` cmdlet with the filter. The following example shows only one of the AMIs returned by the cmdlet; the actual output includes many AMIs.

```
PS C:> Get-EC2Image -Owner amazon, self -Filter $filter_platform  
  
ImageId           : ami-7527031c  
ImageLocation     : amazon/Windows_Server-2012-RTM-English-64Bit-Base-  
2013.11.13  
State             : available  
OwnerId           : 801119661308  
Public            : True  
ProductCodes      : {}  
Architecture      : x86_64  
ImageType         : machine  
KernelId          :  
RamdiskId         :  
Platform          : Windows  
StateReason       :  
ImageOwnerAlias   : amazon  
Name              : Windows_Server-2012-RTM-English-64Bit-Base-2013.11.13  
Description       : Microsoft Windows Server 2012 RTM 64-bit Locale English
```

```
Base AMI provided by Amazon
RootDeviceType      : ebs
RootDeviceName      : /dev/sda1
BlockDeviceMappings : {, ephemeral0, ephemeral1, ephemeral2...}
VirtualizationType  : hvm
Tags                 : {}
Hypervisor           : xen
```

Get-EC2ImageByName

The Get-EC2ImageByName cmdlet enables you to filter the list of Windows AMIs based on the type of server configuration you are interested in.

When run with no parameters, the cmdlet emits the complete set of current filter names for stock Windows images:

```
PS C:> Get-EC2ImageByName

WINDOWS_2012_BASE
WINDOWS_2012_SQL_SERVER_EXPRESS_2012
WINDOWS_2012_SQL_SERVER_STANDARD_2012
WINDOWS_2012_SQL_SERVER_WEB_2012
WINDOWS_2012_SQL_SERVER_EXPRESS_2008
WINDOWS_2012_SQL_SERVER_STANDARD_2008
WINDOWS_2012_SQL_SERVER_WEB_2008
WINDOWS_2008_BASE
WINDOWS_2008_SQL_SERVER_EXPRESS_2012
WINDOWS_2008_SQL_SERVER_STANDARD_2012
WINDOWS_2008_SQL_SERVER_WEB_2012
WINDOWS_2008_SQL_SERVER_EXPRESS_2008
WINDOWS_2008_SQL_SERVER_STANDARD_2008
WINDOWS_2008_SQL_SERVER_WEB_2008
WINDOWS_2008_BEANSTALK_IIS75
WINDOWS_2012_BEANSTALK_IIS8
VPC_NAT
```

You can narrow the set of images returned by specifying filter names using the Names parameter.

```
PS C:> Get-EC2ImageByName -Names WINDOWS_2012_SQL_SERVER_STANDARD_2012

ImageId           : ami-1320047a
ImageLocation      : amazon/Windows_Server-2012-RTM-English-64Bit-
SQL_2012_SP1_Standard-2013.11.13
State              : available
OwnerId            : 801119661308
Public             : True
ProductCodes       : {}
Architecture       : x86_64
ImageType          : machine
KernelId           :
RamdiskId          :
Platform           : Windows
StateReason        :
ImageOwnerAlias     : amazon
```

```
Name                : Windows_Server-2012-RTM-English-64Bit-SQL_2012_SP1_Standard-2013.11.13
Description         : Microsoft Windows Server 2012 RTM 64-bit Locale English with SQL 2012 Standard AMI provided by Amazon
RootDeviceType     : ebs
RootDeviceName      : /dev/sda1
BlockDeviceMappings : {, ephemeral0, ephemeral1, ephemeral2...}
VirtualizationType  : hvm
Tags                : {}
Hypervisor          : xen
```

Launch an Amazon EC2 Instance Using Windows PowerShell

To launch an Amazon EC2 instance, you need the key pair and security group that you created. You also need the ID of an Amazon Machine Image (AMI). For more information, see the following documentation:

- [Create an Amazon EC2 Key Pair Using Windows PowerShell \(p. 20\)](#)
- [Create a Security Group Using Windows PowerShell \(p. 22\)](#)
- [Find an Amazon Machine Image Using Windows PowerShell \(p. 26\)](#)

If you launch an instance that is not within the Free Usage Tier, you are billed after you launch the instance and charged for the time that the instance is running even if it remains idle.

Topics

- [Launching an Instance in EC2-Classical \(p. 28\)](#)
- [Launching an Instance in a VPC \(p. 29\)](#)
- [Launching a Spot Instance in a VPC \(p. 31\)](#)

Launching an Instance in EC2-Classical

The following command creates a single t1.micro instance.

```
PS C:> New-EC2Instance -ImageId ami-7527031c -MinCount 1 -MaxCount 1 -KeyName myPSKeyPair -SecurityGroups myPSSecurityGroup -InstanceType t1.micro

ReservationId      : r-b70a0ef1
OwnerId            : 123456789012
RequesterId        :
Groups             : {myPSSecurityGroup}
GroupName          : {myPSSecurityGroup}
Instances          : {}
```

Your instance is in the pending state initially, but will be in the running state in a few minutes. To view information about your instance, use the Get-EC2Instance cmdlet. If you have more than one instance, you can filter the results on the reservation ID using the Filter parameter. First, create an object of type Amazon.EC2.Model.Filter.

```
PS C:> $reservation = New-Object 'collections.generic.list[string]'
PS C:> $reservation.add("r-b70a0ef1")
```

```
PS C:> $filter_reservation = New-Object Amazon.EC2.Model.Filter -Property @{Name  
= "reservation-id"; Values = $reservation}
```

Next, call Get-EC2Instance with the filter and view the Instances property.

```
PS C:> (Get-EC2Instance -Filter $filter_reservation).Instances  
  
InstanceId           : i-5203422c  
ImageId              : ami-7527031c  
State                : Amazon.EC2.Model.InstanceState  
PrivateDnsName       : ip-10-251-50-12.ec2.internal  
PublicDnsName        : ec2-198-51-100-245.compute-1.amazonaws.com  
StateTransitionReason :  
KeyName              : myPSKeyPair  
AmiLaunchIndex       : 0  
ProductCodes         : {}  
InstanceType         : t1.micro  
LaunchTime           : 12/11/2013 6:47:22 AM  
Placement            : Amazon.EC2.Model.Placement  
KernelId             :  
RamdiskId            :  
Platform             : Windows  
Monitoring           : Amazon.EC2.Model.Monitoring  
SubnetId             :  
VpcId                :  
PrivateIpAddress     : 10.251.50.12  
PublicIpAddress      : 198.51.100.245  
StateReason          :  
Architecture         : x86_64  
RootDeviceType       : ebs  
RootDeviceName       : /dev/sda1  
BlockDeviceMappings  : {/dev/sda1}  
VirtualizationType   : hvm  
InstanceLifecycle    :  
SpotInstanceRequestId :  
License              :  
ClientToken          :  
Tags                 : {}  
SecurityGroups       : {myPSSecurityGroup}  
SourceDestCheck      : False  
Hypervisor            : xen  
NetworkInterfaces    : {}  
IamInstanceProfile   :  
EbsOptimized         : False
```

Launching an Instance in a VPC

The following command creates a single m1.small instance in the specified private subnet. The security group must be one you created for the VPC that contains the specified subnet.

```
PS C:> New-EC2Instance -ImageId ami-7527031c -MinCount 1 -MaxCount 1 -KeyName  
myPSKeyPair -SecurityGroupId sg-5d293231 -InstanceType m1.small -SubnetId subnet-  
d60013bf  
  
ReservationId       : r-b70a0ef1
```

```
OwnerId          : 123456789012
RequesterId      :
Groups           : {}
GroupName        : {}
Instances        : {}
```

Your instance is in the pending state initially, but will be in the running state in a few minutes. To view information about your instance, use the `Get-EC2Instance` cmdlet. If you have more than one instance, you can filter the results on the reservation ID using the `Filter` parameter. First, create an object of type `Amazon.EC2.Model.Filter`.

```
PS C:> $reservation = New-Object 'collections.generic.list[string]'
PS C:> $reservation.add("r-b70a0ef1")
PS C:> $filter_reservation = New-Object Amazon.EC2.Model.Filter -Property @{Name
    = "reservation-id"; Values = $reservation}
```

Next, call `Get-EC2Instance` with the filter and view the `Instances` property.

```
PS C:> (Get-EC2Instance -Filter $filter_reservation).Instances

InstanceId       : i-5203422c
ImageId          : ami-7527031c
State           : Amazon.EC2.Model.InstanceState
PrivateDnsName   : ip-10-25-1-219.ec2.internal
PublicDnsName    :
StateTransitionReason :
KeyName          : myPSKeyPair
AmiLaunchIndex   : 0
ProductCodes     : {}
InstanceType     : m1.small
LaunchTime       : 12/11/2013 6:47:22 AM
Placement        : Amazon.EC2.Model.Placement
KernelId        :
RamdiskId        :
Platform         : Windows
Monitoring       : Amazon.EC2.Model.Monitoring
SubnetId         : subnet-b61f49f0
VpcId            : vpc-a01106c2
PrivateIpAddress : 10.25.1.219
PublicIpAddress  :
StateReason      :
Architecture     : x86_64
RootDeviceType   : ebs
RootDeviceName   : /dev/sda1
BlockDeviceMappings : {/dev/sda1}
VirtualizationType : hvm
InstanceLifecycle :
SpotInstanceRequestId :
License          :
ClientToken      :
Tags             : {}
SecurityGroups   : {myPSSecurityGroup}
SourceDestCheck   : False
Hypervisor        : xen
NetworkInterfaces : {ip-10-25-1-219.ec2.internal}
```

```
IamInstanceProfile      :  
EbsOptimized           : False
```

Launching a Spot Instance in a VPC

The following command requests a Spot Instance in the specified subnet. The security group must be one you created for the VPC that contains the specified subnet.

```
PS C:> $interface1 = New-Object Amazon.EC2.Model.InstanceNetworkInterfaceSpecification  
PS C:> $interface1.DeviceIndex = 0  
PS C:> $interface1.SubnetId = "subnet-b61f49f0"  
PS C:> $interface1.PrivateIpAddress = "10.0.1.5"  
PS C:> $interface1.Groups.Add("sg-5d293231")  
PS C:> Request-EC2SpotInstance -SpotPrice 0.007 -InstanceCount 1 -Type one-time  
-LaunchSpecification_ImageId ami-7527031c -LaunchSpecification_InstanceType  
m1.small -Region us-west-2 -LaunchSpecification_NetworkInterfaces $interface1
```

Amazon S3 from the AWS Tools for Windows PowerShell

In this section, we create a static website from PowerShell using Amazon S3 and CloudFront. In the process, we demonstrate a number of common tasks with these services. This walkthrough is modeled after the Getting Started Guide for [AWS Static Website Hosting](#), which describes a similar process using the [AWS Management Console](#).

The commands shown here assume that you have set default credentials and a default region for your PowerShell session. Therefore, credentials and regions are not included in the invocation of the cmdlets.

See Also

- [Using the AWS Tools for Windows PowerShell \(p. 19\)](#)
- [Hosting Websites on Amazon S3](#)
- [Amazon S3 Console](#)

Topics

- [Create an Amazon S3 Bucket, Verify Its Region, and Optionally Remove It \(p. 31\)](#)
- [Configure an Amazon S3 Bucket as a Website and Enable Logging \(p. 32\)](#)
- [Upload Objects to an Amazon S3 Bucket \(p. 33\)](#)
- [Delete Amazon S3 Objects and Buckets \(p. 34\)](#)
- [Upload In-Line Text Content to Amazon S3 \(p. 35\)](#)

Create an Amazon S3 Bucket, Verify Its Region, and Optionally Remove It

Use the `New-S3Bucket` cmdlet to create a new Amazon S3 bucket. The following examples create a bucket named `website-example`. The name of the bucket must be unique across all regions. The example creates the bucket in the `us-west-1` region.

```
PS C:> New-S3Bucket -BucketName website-example -Region us-west-1

BucketName                               CreationDate
-----
website-example                          Mon, 26 Nov 2012 00:41:08
GMT
```

You can verify the region in which the bucket is located using the `Get-S3BucketLocation` cmdlet.

```
PS C:> Get-S3BucketLocation -BucketName website-example
us-west-1
```

You could use the following line to remove this bucket. We suggest that you leave this bucket in place as we use it in subsequent examples.

```
Remove-S3Bucket -BucketName website-example
```

Note that the bucket-removal process takes some time to complete fully. If you immediately try to create a same-named bucket, the `New-S3Bucket` cmdlet may fail for a period of time.

See Also

- [Using the AWS Tools for Windows PowerShell \(p. 19\)](#)
- [Put Bucket \(Amazon S3 Service Reference\)](#)
- [AWS PowerShell Regions for Amazon S3](#)

Configure an Amazon S3 Bucket as a Website and Enable Logging

Use the `Write-S3BucketWebsite` cmdlet to configure an Amazon S3 bucket as a static website. The following example specifies a name of `index.html` for the default content web page and a name of `error.html` for the default error web page. Note that this cmdlet does not create those pages. They need to be [uploaded as Amazon S3 objects \(p. 33\)](#).

```
PS C:> Write-S3BucketWebsite -BucketName website-example -WebsiteConfiguration_In
dexDocumentSuffix index.html -WebsiteConfiguration_ErrorDocument error.html

RequestId      : A1813E27995FFDDD
AmazonId2      : T7hlD0eLqA5Q2XfTe8j2q3SLoP3/5XwhUU3RyJBGHU/LnC+CIWLeGgP0MY24xAlI
ResponseStream :
Headers        : {x-amz-id-2, x-amz-request-id, Content-Length, Date...}
Metadata       : {}
ResponseXml    :
```

See Also

- [Using the AWS Tools for Windows PowerShell \(p. 19\)](#)
- [Put Bucket Website \(Amazon S3 API Reference\)](#)
- [Put Bucket ACL \(Amazon S3 API Reference\)](#)

Upload Objects to an Amazon S3 Bucket

Use the `Write-S3Object` cmdlet to upload files from your local file system to an Amazon S3 bucket as objects. The example below creates and uploads two simple HTML files to an Amazon S3 bucket, and verifies the existence of the uploaded objects. The `-File` parameter to `Write-S3Object` specifies the name of the file in the local file system. The `-Key` parameter specifies the name that the corresponding object will have in Amazon S3.

Amazon infers the content-type of the objects from the file extensions, in this case, ".html".

```
PS C:> # Create the two files using here-strings and the Set-Content cmdlet
PS C:> $index_html = @"
>> <html>
>>   <body>
>>     <p>
>>       Hello, World!
>>     </p>
>>   </body>
>> </html>
>> "@
>>
PS C:> $index_html | Set-Content index.html
PS C:> $error_html = @"
>> <html>
>>   <body>
>>     <p>
>>       This is an error page.
>>     </p>
>>   </body>
>> </html>
>> "@
>>
PS C:> $error_html | Set-Content error.html
PS C:> # Upload the files to Amazon S3 using a foreach loop
PS C:> foreach ($f in "index.html", "error.html") {
>> Write-S3Object -BucketName website-example -File $f -Key $f -CannedACLName
PublicRead
>> }
>>
PS C:> # Verify that the files were uploaded
PS C:> Get-S3BucketWebsite -BucketName website-example

IndexDocumentSuffix                                ErrorDocument
-----
index.html                                           error.html
```

Canned ACL Options

The values for specifying canned ACLs in PowerShell Tools is the same as that used by the AWS SDK for .NET. Note, however, that these are different from the values used by the Amazon S3 `Put Object` action. AWS Tools for Windows PowerShell supports the following canned ACLs:

- NoACL
- Private
- PublicRead
- PublicReadWrite

- AuthenticatedRead
- BucketOwnerRead
- BucketOwnerFullControl
- LogDeliveryWrite

Note Regarding Multipart Upload

If you use the Amazon S3 API, to upload a file that is larger than 5 GB in size, you need to use multipart upload. However, the `Write-S3Object` cmdlet provided by the PowerShell Tools is able to transparently handle file uploads that are greater than 5 GB.

Test the Website

At this point, you can test the website by navigating to it using a browser. URLs for static websites hosted in Amazon S3 follow a standard format.

```
http://<bucket-name>.s3-website-<region>.amazonaws.com
```

For example:

```
http://website-example.s3-website-us-west-1.amazonaws.com
```

See Also

- [Using the AWS Tools for Windows PowerShell \(p. 19\)](#)
- [Put Object \(Amazon S3 API Reference\)](#)
- [Canned ACLs \(Amazon S3 API Reference\)](#)

Delete Amazon S3 Objects and Buckets

This section describes how to delete the website that was created in the preceding sections. This process involves simply deleting the objects for the HTML files and then deleting the Amazon S3 bucket for the site itself.

Use the `Remove-S3Object` cmdlet to delete the objects for the HTML files from the Amazon S3 bucket.

```
PS C:> foreach ( $obj in "index.html", "error.html" ) {  
>> Remove-S3Object -BucketName website-example -Key $obj  
>> }  
>>  
IsDeleteMarker  
-----  
False
```

The `False` response is an expected artifact of the way that Amazon S3 processes the request. In this context, it does not indicate an issue.

Use the `Remove-S3Bucket` cmdlet to delete the now empty Amazon S3 bucket for the site.

```
PS C:> Remove-S3Bucket -BucketName website-example
```

```
RequestId      : E480ED92A2EC703D
AmazonId2      : k6tqaqC1nMkoeYwbuJXUx1/UDa49BJd6dfLN0Ls1mwYNPHjbc8/Nyvm6AGbwcc2P
ResponseStream :
Headers        : {x-amz-id-2, x-amz-request-id, Date, Server}
Metadata       : {}
ResponseXml     :
```

In version 1.1 and later of the PowerShell Tools, you can specify the `-DeleteObjects` switch to `Remove-Bucket` which causes the cmdlet to first delete all objects and object versions in the specified bucket before attempting to remove the bucket itself. Depending on the number of objects/object versions in the bucket, this operation can take a substantial amount of time to complete. In versions of the PowerShell Tools prior to v1.1, the bucket had to be empty in order for `Remove-Bucket` to delete it.

Note that unless `-Force` is specified, you will be prompted for confirmation before the cmdlet runs.

See Also

- [Using the AWS Tools for Windows PowerShell \(p. 19\)](#)
- [Delete Object \(Amazon S3 API Reference\)](#)
- [DeleteBucket \(Amazon S3 API Reference\)](#)

Upload In-Line Text Content to Amazon S3

The `Write-S3Object` cmdlet supports the ability to upload in-line text content to Amazon S3. Using the `-Content` parameter (alias `-Text`), you can specify text-based content that should be uploaded to Amazon S3 without needing to place it into a file first. The parameter accepts simple one-line strings as well as here strings that contain multiple lines.

```
# Specifying content in-line, single line text:
write-s3object mybucket -key myobject.txt -content "file content"

# Specifying content in-line, multi-line text: (note final newline needed to
end in-line here-string)
write-s3object mybucket -key myobject.txt -content @"
>> line 1
>> line 2
>> line 3
>> "@
>>

# Specifying content from a variable: (note final newline needed to end in-line
here-string)
$x = @"
>> line 1
>> line 2
>> line 3
>> "@
>>
write-s3object mybucket -key myobject.txt -content $x
```

AWS Identity and Access Management from the AWS Tools for Windows PowerShell

This section describes some common tasks related to AWS Identity and Access Management (IAM) and how to perform them using the AWS Tools for Windows PowerShell.

The commands shown here assume that you have set default credentials and a default region for your PowerShell session. Therefore, credentials and regions are not included in the invocation of the cmdlets.

Topics

- [Create New IAM Users and Groups \(p. 36\)](#)
- [Set an IAM Policy for an IAM User \(p. 37\)](#)
- [Set an Initial Password for an IAM User \(p. 38\)](#)
- [Create Security Credentials for an IAM User \(p. 38\)](#)

Create New IAM Users and Groups

This section describes how to create a new IAM Group and a new IAM User and then add the user to the group.

First, use the `New-IAMGroup` cmdlet to create the group. Although we've included it here, the `-Path` parameter is optional.

```
PS C:> New-IAMGroup -Path "/ps-created-groups/" -GroupName "powerUsers"

Path          : /ps-created-groups/
GroupName     : powerUsers
GroupId       : AGPAJPHUEYD5XPCGIUH3E
Arn           : arn:aws:iam::455364113843:group/ps-created-groups/powerUsers
CreateDate    : 11/20/2012 3:32:50 PM
```

Next, use the `New-IAMUser` cmdlet to create the user. Similar to the preceding example, the `-Path` parameter is optional.

```
PS C:> New-IAMUser -Path "/ps-created-users/" -UserName "myNewUser"

Path          : /ps-created-users/
UserName      : myNewUser
UserId        : AIDAJ0JSPSPXADHBT7IN6
Arn           : arn:aws:iam::455364113843:user/ps-created-users/myNewUser
CreateDate    : 11/20/2012 3:26:31 PM
```

Finally, use the `Add-IAMUserToGroup` cmdlet to add the user to the group.

```
PS C:> Add-IAMUserToGroup -UserName myNewUser -GroupName powerUsers

ServiceResponse
-----
Amazon.IdentityManagement.Model.AddUserToGroupResponse
```

To verify that the `powerUsers` group contains the `myNewUser`, use the `Get-IAMGroup` cmdlet.

```
PS C:> Get-IAMGroup -GroupName powerUsers
```

Group	Marker	Users	IsTruncated
-----	-----	-----	-----
Amazon.IdentityManagement...		{myNewUser}	False

You can also view IAM users and groups with the AWS Management Console

<https://console.aws.amazon.com/iam/home?s=Users> [Users View]

<https://console.aws.amazon.com/iam/home?s=Groups> [Groups View]

See Also

- [Using the AWS Tools for Windows PowerShell \(p. 19\)](#)
- [Adding a New User to Your AWS Account \(IAM User Guide\)](#)
- [CreateGroup \(IAM Service Reference\)](#)

Set an IAM Policy for an IAM User

The following commands show how to assign an IAM policy to an IAM user. The policy specified below provides the user with "Power User Access". This policy is identical to the **Power User Access** policy template provided in the IAM console. The name for the policy shown below follows the naming convention used for IAM policy templates such as the template for **Power User Access**. The convention is

```
<template name>+<user name>+<date stamp>
```

In order to specify the policy document, we use a PowerShell here-string. We assign the contents of the here-string to a variable and then use the variable as a parameter value in `Write-IAMUserPolicy`.

```
PS C:> $policyDoc = @"
>> {
>>   "Version": "2012-10-17",
>>   "Statement": [
>>     {
>>       "Effect": "Allow",
>>       "NotAction": "iam:*",
>>       "Resource": "*"
>>     }
>>   ]
>> }
>> @"

PS C:> Write-IAMUserPolicy -UserName myNewUser -PolicyName "PowerUserAccess-
myNewUser-201211201605" -PolicyDocument $policyDoc

ServiceResponse
```

```
-----  
Amazon.IdentityManagement.Model.PutUserPolicyResponse
```

See Also

- [Using the AWS Tools for Windows PowerShell \(p. 19\)](#)
- [Using Windows PowerShell "Here-Strings"](#)
- [PutUserPolicy](#)

Set an Initial Password for an IAM User

The following example demonstrates how to use the `New-IAMLoginProfile` cmdlet to set an initial password for an IAM user.

```
PS C:> New-IAMLoginProfile -UserName myNewUser -Password "&!123!&"  
  
UserName                                CreateDate  
-----                                -  
myNewUser                                11/20/2012 4:23:05 PM
```

Use the `Update-IAMLoginProfile` cmdlet to update the password for an IAM user.

See Also

- [Using the AWS Tools for Windows PowerShell \(p. 19\)](#)
- [Managing Passwords](#)
- [CreateLoginProfile](#)

Create Security Credentials for an IAM User

The following example uses the `New-IAMAccessKey` cmdlet to create security credentials for an IAM user. A set of security credentials comprises an Access Key ID and a Secret Key. Note that an IAM user can have no more than two sets of credentials at any given time. If you attempt to create a third set, the `New-IAMAccessKey` cmdlet returns an error.

```
PS C:> New-IAMAccessKey -UserName myNewUser  
  
UserName      : myNewUser  
AccessKeyId   : AKIAIOSFODNN7EXAMPLE  
Status        : Active  
SecretAccessKey : wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKE  
CreateDate    : 11/20/2012 4:30:04 PM
```

Use the `Remove-IAMAccessKey` cmdlet to delete a set of credentials for an IAM user. Specify credentials to delete using the Access Key ID.

```
PS C:> Remove-IAMAccessKey -UserName myNewUser -AccessKeyId AKIAIOSFODNN7EXAMPLE
```

```
ServiceResponse
-----
Amazon.IdentityManagement.Model.DeleteAccessKeyResponse
```

Using the AWS Tools for Windows PowerShell with Amazon SQS and Amazon SNS

This section provides instructions to:

- Create an Amazon SQS queue and get queue ARN (Amazon Resource Name).
- Create an Amazon SNS topic.
- Give permissions to the SNS topic so that it can send messages to the queue.
- Subscribe the queue to the SNS topic
- Give IAM users or AWS accounts permissions to publish to the SNS topic and read messages from the SQS queue.
- Verify results by publishing a message to the topic and reading the message from the queue.

Create an Amazon SQS queue and get queue ARN

The following command creates an SQS queue:

```
New-SQSQueue -QueueName MyQueue -Region us-west-2
```

The URL of the created queue is returned:

```
https://sqs.us-west-2.amazonaws.com/123456789012/MyQueue
```

The following command gets the queue ARN

```
Get-SQSQueueAttribute -QueueUrl https://sqs.us-west-2.amazonaws.com/123456789012/MyQueue -AttributeName QueueArn
```

The ARN of the created queue is returned:

```
...
QueueARN : arn:aws:sqs:us-west-2:123456789012:MyQueue
...
```

Create an Amazon SNS topic

The following command creates an SNS queue:

```
New-SNSTopic -Name MyTopic -Region us-west-2
```

The ARN of the created topic is returned:

```
arn:aws:sns:us-west-2:123456789012:MyTopic
```

Give permissions to the SNS topic

The following command gives permissions to the SNS topic so that it can send messages to the queue:

```
# create the queue and topic to be associated
$url = New-SQSQueue -QueueName "myQueue"
$topicarn = New-SNSTopic -Name "myTopic"

# get the queue ARN to inject into the policy; it will be returned
# in the output's QueueARN member but we need to put it into a variable
# so text expansion in the policy string takes effect
$qarn = (Get-SQSQueueAttribute -QueueUrl $url -AttributeNames
"QueueArn").QueueARN

# construct the policy and inject arns
$policy = @"
{
  "Version": "2008-10-17",
  "Id": "$qarn/SQSPOLICY",
  "Statement": [
    {
      "Sid": "1",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "SQS:SendMessage",
      "Resource": "$qarn",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "$topicarn"
        }
      }
    }
  ]
}
"@

# set the policy
Set-SQSQueueAttribute -QueueUrl $url -Attribute @{ Policy=$policy }
```

The following is returned:

```
ServiceResponse
-----
<?xml version="1.0" encoding="utf-16"?>...
```

Subscribe the queue to the SNS topic

The following command subscribes the queue *MyQueue* to the SNS topic *MyTopic*:

```
Connect-SNSNotification -TopicArn arn:aws:sns:us-west-2:123456789012:MyTopic -
Protocol SQS -Endpoint arn:aws:sqs:us-west-2:123456789012:MyQueue -Region us-
west-2
```

The Subscription Id is returned:

```
arn:aws:sns:us-west-2:123456789012:ps-cmdlet-topic:f8ff77c6-e719-4d70-8e5c-
a54d41feb754
```

Give permissions

The following command gives permission to perform the `sns:Publish` action on the topic *MyTopic*

```
Add-SNSPermission -TopicArn arn:aws:sns:us-west-2:123456789012:MyTopic -Label
ps-cmdlet-topic -AWSAccountIds 123456789012 -ActionNames publish -Region us-
west-2
```

The following is returned:

```
ServiceResponse
-----
<?xml version="1.0" encoding="utf-16"?>...
```

The following command gives permission to perform the `sqs:ReceiveMessage` and `sqs:DeleteMessage` actions on the queue *MyQueue*

```
Add-SQSPermission -QueueUrl https://sqs.us-west-2.amazon
aws.com/123456789012/MyQueue -Region US-West-2 -AWSAccountId "1234-5678-9012"
-Label queue-permission -ActionName SendMessage, ReceiveMessage
```

The following is returned:

```
ServiceResponse
-----
<?xml version="1.0" encoding="utf-16"?>...
```

Verify results

The following command publishes a message to the SNS topic *MyTopic*

```
Publish-SNSMessage -TopicArn arn:aws:sns:us-west-2:123456789012:MyTopic -Message
"Have A Nice Day!" -Region us-west-2
```

The MessageId is returned:

```
4914beb6-f8d2-5568-989f-f7909cefab79
```

The following command retrieves the message from the SQS queue *MyQueue*

```
Receive-SQSMessage -QueueUrl https://sqs.us-west-2.amazon
aws.com/123456789012/MyQueue -Region us-west-2
```

The following is returned:

```
MessageId      : 03204f1d-1d65-4733-9eed-fc9cd514873a
ReceiptHandle  : uUk89DYFzt3SjcTmtVq9VLAxpcJU5hH0KkInt+Hq6AxnWLG1lEg1RLnPlIrkr
fLNmujk8+p2HrTCw0+1nLHAA+rfcy0m0f7Hxvm9iGR
                WMcFcCp4woccvY
wQJW/if62D8R14v4JtSlTEiY2ukxl/Zb4xqC9WN3+M0YZ/HW1euFb/tIE0qLQnKcOy
oQ4Hj1d5Wgc/IFo0cYNvOuM
                x8pRxey0HKpah80TrFiQFcCXbMKiuTqOI6yceInyAJ8YWwfKp
jatc2zUcq5PqcrYMTbs4jK/zJc4uVhZNMUMCu2fA5EM4=
MD5OfBody     : 60509281ad1bfd6980e84f9d64bbf9ab
Body          : {
                "Type" : "Notification",
                "MessageId" : "4914beb6-f8d2-5568-989f-f7909cefab79",
                "TopicArn" : "arn:aws:sns:us-west-2:803981987763:MyTopic",
                "Message" : "Have A Nice Day!",
                "Timestamp" : "2012-11-21T05:09:17.905Z",
                "SignatureVersion" : "1",
                "Signature" : "GpF4Dhb5GotbtK883ccm1s59+7vnZMdcjxrAVYU7+igD
FVWrvI6/bDfws5GcjT/IP9GxG6UJ55b8pu1+jzujaN
                YhZpr52mJfQHGRtM8FN0IacCDDRQ00tXCHl0a6GP1s7RVIUNgCOzR/tbCCpJo1
Gace+j0F1uf26LN4453RR6o=",
                "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleN
otificationService-f3ecfb7224c7233fe7b
                b5f59f96de52f.pem",
                "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Ac
tion=Unsubscribe&SubscriptionArn=arn:aws:s
                ns:us-west-2:803981987763:ps-cmdlet-topic:f8ff77c6-e719-4d70-8e5c-
a54d41feb754"
                }
Attribute     : {}
```

CloudWatch from the AWS Tools for Windows PowerShell

This section shows an example of using the PowerShell Tools to publish custom metric data to CloudWatch.

This example assumes that you have set default credentials and a default region for your PowerShell session. Therefore, credentials and a region are not included in the invocation of the cmdlets.

Publish a Custom Metric to Your CloudWatch Dashboard

The PowerShell code below initializes an CloudWatch MetricDatum object and posts it to the service. You can see the result of this operation by navigating to the [CloudWatch console](#).

```
$dat = New-Object Amazon.CloudWatch.Model.MetricDatum
$dat.Timestamp = (Get-Date).ToUniversalTime()
```

```
$dat.MetricName = "New Posts"  
$dat.Unit = "Count"  
$dat.Value = ".50"  
Write-CWMetricData -Namespace "Usage Metrics" -MetricData $dat
```

Please note the following:

- The date-time information used to initialize `$dat.Timestamp` needs to be in Universal Time (UTC).
- The value used to initialize `$dat.Value` can be either a string value enclosed in quotes or a numeric value (no quotes). A string value is shown above.

See Also

- [Using the AWS Tools for Windows PowerShell \(p. 19\)](#)
- [AmazonCloudWatch.PutMetricData \(.NET SDK Reference\)](#)
- [MetricDatum \(Service API Reference\)](#)
- [Amazon CloudWatch Console](#)

See Also

- [Getting Started with the AWS Tools for Windows PowerShell \(p. 8\)](#)

Document History

This topic describes significant changes to the documentation for the AWS Tools for Windows PowerShell.

Last documentation update: May 15, 2013

We also update the documentation periodically in response to customer feedback. To send us feedback, use the feedback link--"Did this page help you?"--located at the top and bottom of each page.

For additional information about changes and updates to the PowerShell Tools, see the [release notes](#).

AWS Tools for Windows PowerShell v1.1.1.0

Release Date: 2013-05-15

Summary of Changes

- Collection output from cmdlets is always enumerated to the PowerShell pipeline
- Automatic support for pageable service calls
- New \$AWSHistory shell variable collects service responses and optionally service requests
- AWSRegion instances use Region field instead of SystemName to aid pipelining
- Remove-S3Bucket supports a -DeleteObjects switch option
- Fixed usability issue with Set-AWSCredentials
- Initialize-AWSDefaults reports from where it obtained credentials and region data
- Stop-EC2Instance accepts Amazon.EC2.Model.Reservation instances as input
- Generic List<T> parameter types replaced with array types (T[])
- Cmdlets that delete or terminate resources prompt for confirmation prior to deletion
- Write-S3Object supports in-line text content to upload to Amazon S3

AWS Tools for Windows PowerShell v1.0.1.0

Release Date: 2012-12-21

The install location of the PowerShell Tools module has changed so that environments using Windows PowerShell version 3 can take advantage of auto-loading.

- The module and supporting files are now installed to an *AWSPowerShell* subfolder beneath *AWS Tools\PowerShell*. Files from previous versions that exist in the *AWS Tools\PowerShell* folder are automatically removed by the installer.
- The `PSModulePath` for Windows PowerShell (all versions) is updated in this release to contain the parent folder of the module (*AWS Tools\PowerShell*).
- For systems with Windows PowerShell version 2, the Start Menu shortcut **Amazon Web Services\Windows PowerShell for AWS** is updated to import the module from the new location and then run `Initialize-AWSDefaults`.
- For systems with Windows PowerShell version 3, the Start Menu shortcut **Amazon Web Services\Windows PowerShell for AWS** is updated to remove the `Import-Module` command, leaving just `Initialize-AWSDefaults`.
- If you edited your PowerShell profile to perform an `Import-Module` of the *AWSPowerShell.psd1* file, you will need to update it to point to the file's new location (or, if using PowerShell version 3, remove the `Import-Module` statement as it is no longer needed).

As a result of these changes, the PowerShell Tools module is now listed as an available module when executing `Get-Module -ListAvailable`. In addition, for users of Windows PowerShell version 3, the execution of any cmdlet exported by the module will automatically load the module in the current PowerShell shell without needing to use `Import-Module` first. This enables interactive use of the cmdlets on a system with an execution policy that disallows script execution.

AWS Tools for Windows PowerShell v1.0.0.0

Release Date: 2012-12-06

Initial release